# SoundMex Documentation

## Sound-Toolkit for MatLab

*User Manual*

Copyright HörTech gGmbH, Marie-Curie-Str. 2, D-26129 Oldenburg, Germany

http://www.hoertech.de

No warranty, subject to alteration

# License agreement

IMPORTANT- PLEASE READ CAREFULLY:

BY INSTALLING THE SOFTWARE (AS DEFINED BELOW), COPYING THE SOFTWARE AND/OR CLICKING ON THE "ACCEPT" BUTTON BELOW, YOU (EITHER ON BEHALF OF YOURSELF AS AN INDIVIDUAL OR ON BEHALF OF AN ENTITY AS ITS AUTHORIZED REPRESENTATIVE) AGREE TO ALL OF THE TERMS OF THIS END USER LICENSE AGREEMENT ("AGREEMENT") REGARDING YOUR USE OF THE SOFTWARE. IF YOU DO NOT AGREE WITH ALL OF THE TERMS OF THIS AGREEMENT, DO NOT INSTALL AND/OR USE THE SOFTWARE.

## DEFINITIONS

The term "Software" includes all software distributed with this License including all documentation. The "Software" is licensed to you under the terms specified in the License Grant below.

## HIGH RISK ACTIVITIES

The Software is not fault-tolerant and is not designed, manufactured or intended for use as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or other medical devices, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). HörTech gGmbH and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

## OWNERSHIP AND COPYRIGHT

This Software is owned by HörTech gGmbH or its suppliers and is protected by copyright law and international copyright treaty. Therefore you must treat this Software like any other copyrighted material. You acknowledge that no title to the intellectual property in the Software is transferred to you. Title, ownership, rights, and intellectual property rights in and to the Software shall remain in HörTech gGmbH.

## LICENSE GRANT

Subject to the license terms, HörTech gGmbH hereby grants you a non-exclusive, non-transferable (except under the terms below) license to install and to use the Software under the terms of this license. Except as provided in this license agreement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share or electronically transmit the Software. You may only either make one copy of the Software solely for backup or archival purposes or transfer the Software to a single hard disk provided you keep the original solely for backup or archival purposes. you agree not to modify the Software or attempt to decipher, de-compile, disassemble or reverse engineer the Software, except to the extent applicable laws specifically prohibit such restriction.

## LICENSE TRANSFER

You may transfer your license and the rights granted in the license to a third party only if a) the third party agrees to this license agreement, b) you completely uninstall and delete all copies of this Software, c) all parts of the Software and its distribution are transfered to the third party and d) the transfer includes the current version and all prior versions of the Software.

## DISCLAIMER OF WARRANTY

THIS SOFTWARE IS SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF NONINFRINGEMENT OF THIRD PARTY RIGHTS, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. GOOD DATA PROCESSING PROCEDURE DICTATES THAT ANY PROGRAM BE THOROUGHLY TESTED WITH NON-CRITICAL DATA BEFORE RELYING ON IT. THE USER MUST ASSUME THE ENTIRE RISK OF USING THE PROGRAM. ANY LIABILITY OF THE SELLER WILL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT OR REFUND OF PURCHASE PRICE. Under and restricted by the above terms, HörTech gGmbH warrants that the Software, as updated and when properly used, will perform substantially in accordance with its accompanying documentation, and the Software media will be free from defects in materials and workmanship. The limited warranty is void if the Software fails as a result of accident, abuse, misapplication or modification. LIMITATION OF LIABILITY You must assume the entire risk of using the Software. IN NO EVENT SHALL HörTech gGmbH BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND ARISING OUT OF THE USE OF THE HörTech gGmbH's SOFTWARE, EVEN IF HörTech gGmbH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL HörTech gGmbH's LIABILITY FOR ANY CLAIM, WHETHER IN CONTRACT, TORT, OR ANY OTHER THEORY OF LIABILITY, EXCEED THE LICENSE FEE PAID BY YOU. THIS LIMITATION SHALL APPLY TO CLAIMS OF PERSONAL INJURY TO THE EXTENT PERMITTED BY LAW.

# 1 Contents

License agreement....2
1 Contents....4
2 Introduction....5
3 System Requirements....6
4 Installing SoundMex....6
5 Uninstalling SoundMex....7
6 Command Description....8
6.1.1 General Commands....9
6.1.2 Playback Commands....12
6.1.3 Recording Commands....19
6.1.4 Input-Output-Commands....23
6.1.5 Device Commands....24
6.1.6 Ringbuffer Commands....26
6.1.7 Special Commands....28
7 The SoundMex realtime DSP-Plugin-Pipe....31
8 The SoundMex DSP-SDK....33
9 Feature Matrix....34
10 Common problems (FAQ)....36
11 Upcoming Features....36
12 Version History....37

# 2 Introduction

SoundMex is a powerful tool for 16 and 32bit sound applications in MatLab. SoundMex is especially designed for acoustic measurement tasks (Psychoacoustics, Physical Acoustics, Neuro Physiology).

***The software SoundMex is not a medical device or an accessory of a medical device, respectively, and not intended to drive a medical device.***

The main features of SoundMex are:

- synchronous and asynchronous playback and recording
- real-time mixing to asynchronous playback
- sample-accurate sychronization of multiple playback and recording devices (devices themselves must support synchronization!!)
- realtime visualization of levels, time signals and spectra
- ringbuffer mode: generate sound data online (while device is already playing!) and 'feed' new data to device
- record audio while playing and get recorded data directly to MatLab
- record from multiple devices synchroneously to disk (harddisk recording)
- Realtime DSP-Plugin-Pipe with various DSP-plugins for real-time signal processing
- DSP-Plugin-SDK (Software Development Kit) for user defined realtime plugins for blockwise wave and spectrum manipulation
    - script based plugins: user defined MatLab commands are called from the DSP-pipe. Manipulated data are written back to the pipe
    - Simple API for development of fast binary user plugins in any programming language
- direct I/O: recorded data can be directly (or after processing the data with DSP-Plugins) mapped to a waveout for playback
- special command for 'highlighting' buttons on a MatLab window: sample accurate hightlighting at desired playback 'positions' (for signal-synchronous user feedback)

A free version of SoundMex is available with the follwing restrictions:

- only a small subset of commands is available (see feature matrix below)
- only one device can be used at the same time. The first call to a command with a device parameter selects the device to use. A device with another ID can be used after the next call of soundmex(‘init’)

A demo version of SoundMex is available for free. The demo version has the following restrictions:

- a demo message is shown on 'init' and after every five minutes
- sound input and output is stopped every five minutes

- additional sound (the spoken words 'soundmex demo') is added to all playbacks and recordings at random times

# 3 System Requirements

SoundMex runs on MatLab 5.3 and above on Windows 9x and above. The computer should have at least 128 MB of RAM and a processor speed of 500MHz. The more power your computer has, the more tracks, devices and DSP-plugins can be handled simultaneously.

# 4 Installing SoundMex

Run SOUNDMEX.EXE and follow the installation instructions. The following files will be installed on your computer:

| Directory | File(s) | Descritpion |
|---|---|---|
| BIN | SOUNDMEX.DLL, SOUNDDLL.DLL, SOUNDDLL.INI, (SOUNDDLL.LIC) | Main program files + INI-File (see below) + Licence file. Copy these files to any directiory in your Matlab search path (or add the current directory to it) |
| Windows-System-Directory | MMTOOLS_C5.BPL<br>VCL50.BPL<br>VCLX50.BPL<br>VCL50.DE<br>TEE50.BPL<br>TEEUI50.BPL | Runtime libraries. These files are copied to your Windows-system directory. Attention: copying these files to the Matlab search path or to your current working directory will not be sufficient to run SoundMex! |
| EXAMPLES | various | Matlab examples and wave files |
| PLUGINS | various | Plugins shipped with SoundMex |
| DSP-SDK | various | Documentation, examples and sample C-Code for the DSP-Software Development Kit |
| MANUAL | SOUNDMEX.PDF<br>SOUNDMEX PLUGINS.PDF | PDF Help |
| MANUAL/HTML | SOUNDMEX.HTML<br>+ related files | HTML help for access from Matlab. Copy HTML and subfolder 'Soundmex-Files' to your MatLab ref path (see below) |

After installing you should add the BIN directory of your SoundMex installation (e.g. C:\SoundMex\bin) to your MatLab path. If you want to access the HTML-help through the matlab command 'doc soundmex' please copy the file SOUNDMEX.HTML and the subfolder SOUNDMEX-FILES from the 'Manual\HTML' subdirectory (e.g. C:\SoundMex\manual) to the directory

```
<matlab path>\help\techdoc\ref
```

(e.g C:\MATLAB6p5\help\techdoc\ref)

You can set the temporary path of SoundMex (path were recorded data will be stored) by modifying the path variable in the file `SOUNDDLL.INI` (or in your user defined INI-File, see command 'init'), e.g.

```
[System]
TempPath=c:\temp
```

*Caution: Never run SoundMex from network drives or set its temporary path to a network drive!*

# 5 Uninstalling SoundMex

An uninstaller is shipped with SoundMex. Select 'Uninstall SoundMex' from the start menu or run the uninstaller from the control panel.

The copy protection driver will not be removed by this uninstaller to avoid problems with other dongles.

If you want to remove the dongle driver manually, please perform the following steps

If a version of SoundMex v1.18 or below or SoundMex2 2.08 or below was installed on your system please call the command

hinstall -r

from the command prompt in the HASP-subdirectory of your SoundMex installation (e.g. c:\soundmex\hasp), or if this directory does not exist form your system directory (e.g. c:\winnt\system32). After the command has completed the file HINSTALL.EXE can be deleted.

For removing the dongle driver for SoundMey v1.19 or above or SoundMex2 v2.09 or above please call the command

haspdinst -r

from the command prompt in the HASP-subdirectory of your SoundMex installation (e.g. c:\soundmex2\hasp) . After the command has completed the file HASPDINST.EXE can be deleted.

# 6 Command Description

All SoundMex commands have a similar syntax:

```
[outarg1, outarg2,…] = soundmex('command', arg1, arg2,…);
```

All commands return one output argument at least. This return value `outarg1` indicates success or failure of the command itself, where 1 is returned on success and 0 on any error. Evaluate this first return value to react on any error before calling further soundmex commands, e.g.:

```
Success = soundmex('init');
if (~Success)
error('Cannot initialize soundmex!');
end
```

Commands with additional return values write these to outarg[2] … outarg[n], e.g.:

```
[Success, ClipCount] = soundmex('hasclipped');
if (~Success)
error('Error in command hasclipped');
end
disp(['playback has clipped ' num2str(ClipCount) ' times'];
```

```
[Success, NumBuf, BufSiz] = soundmex('getplaybuffer');
if (~Success)
error('error in command getplaybuffer');
end
disp(['playback device has ' num2str(NumBuf) ' buffers of '
num2str(BufSiz) ' bytes size']);
```

The table below shows a list of all available commands sorted by functionality. In the column 'Examples' you can find a list of example scripts that are related to the specified command.

### 6.1.1 General Commands

| Command | Description | Examples |
|---|---|---|
| help | soundmex('help', command)<br>shows help for specified command<br>use soundmex('help') to see a list of all commands<br>use soundmex('help', 'all') to see help on all commands<br>PARAMETERS:<br>command: name of the command or 'all'<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| version | soundmex('version')<br>shows version information about SOUNDMEX.DLL and SOUNDDLL.DLL | |
| init | soundmex('init', IniFile, InitMode)<br>initializes the devices. Must be called before any other action (except 'isinitialized' and 'getdeviceinfo'<br>PARAMETERS:<br>IniFile: optional filename for the inifile to use (especially useful when usingDSP-Plugins)<br>InitMode: optional parameter with the follwing values<br>'free': switches licence mode to 'free' with limited features<br>'force': if soundmex was already initialized, the internal 'exit' command is called, followed by 'init'<br>'forcefree': if soundmex was already initialized, the internal 'exit' command is called, followed by soundmex('init', 'free') | all examples<br>checklicence.m |

| | | |
|---|---|---|
| | RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: string containing licence type information | |
| isinitialized | soundmex('isinitialized')<br>use this command to determine if soundmex is already initialized.<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: 1 if initialized, 0 else | |
| exit | soundmex('exit')<br>resets the devices. Is called automatically when unloading soundmex. Use this command for resetting all devices and buffers.<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| show | soundmex('show', [OUTDEVICES], [INDEVICES])<br>Shows window with devices and data visualization<br>PARAMETERS:<br>[OUTDEVICES]: row vector with device ID's of playback devices to show<br>[INDEVICES]: row vector with device ID's of recording devices to show<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | device_info.m<br>mark_buttons.m<br>sync_play.m |
| hide | soundmex('hide')<br>Hides window with devices<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| getlasterror | soundmex('getlasterror', param)<br>shows last error string<br>RETURN VALUES: | error_handling.m |

| | | |
|---|---|---|
| | outarg[1]: 1 on success, 0 on any error<br>outarg[2]: last error message as string | |
| showerror | soundmex('showerror', param)<br>toggles error string output. Returnvalue remains untouched<br>PARAMETERS:<br>param: 'on' or 'off'<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | error_handling.m |

### 6.1.2 Playback Commands

| Command | Description | Examples |
|---|---|---|
| playfile | soundmex('playfile', filename, Mode, LoopCount, DeviceID, [nBits], [random])<br>plays sound from file.<br>PARAMETERS:<br>filename: the filename! Must be a valid Windows-PCM-Wavefile!<br>Mode: string ('sync' or 'async')<br>async: starts playing file and returns immediately<br>sync: plays file and returns after playback is complete<br>LoopCount: number of times file should be played. 0 does infinite loop (in async mode only)<br>DeviceID: optional WaveOutDeviceID, default is 0<br>nBits: optional bitlength for output. Specify this value if ouput format should be different from file format (16 or 32 only!). Use '32' for 24-bit devices!!<br>ATTENTION: if you want to use the 'nBits' flag you must specify the DeviceID too (not optional any longer!)<br>ATTENTION: What kind of 32bit waves you can use depends on the capabilities of your device!! if you use 'wrong' formats the output may overdrive! To be sure use 32bit-Integer wave files<br>'Goody': if you select 'async' and LoopCount = 0 you can specify an extra parameter 'random'. if so you must give the DeviceID too!!!. if you specify 'random' as last parameter the playback start position within the file is chosen randomly<br>RETURN VALUES: | calculate_while_playing.m<br>mark_buttons.m<br>modulate.m<br>record.m |

| | | |
|---|---|---|
| | outarg[1]: 1 on success, 0 on any error | |
| preparefile | soundmex('preparefile', filename, Mode, LoopCount, DeviceID, [nBits], [random])<br>same as playfile, but device is opened only. Use this for multiple devices to synchronize them and start all with 'playprepared'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>ATTENTION: ONLY ASYNC PLAYBACK MODE SUPPORTED!<br>PARAMETERS:<br>same as in playfile<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | see sync_play.m for preparemem<br>sync_record.m |
| filetofile | soundmex('filetofile', filename, outfile, DeviceID, [nBits])<br>pumps file through DSP-pipe to file.<br>ATTENTION: volumes set by command 'setvolume' are NOT applied!<br>PARAMETERS:<br>filename: the filename! Must be a valid Windows-PCM-Wavefile!<br>outfile: name of the output file<br>DeviceID: optional WaveOutDeviceID, default is 0. Must be set here for plugins!<br>nBits: optional bitlength for output. Specify this value if ouput format should be different from file format (16 or 32 only!) Use '32' for 24-bit devices!!<br>ATTENTION: if you want to use the 'nBits' flag you must specify the DeviceID too (not optional any longer!)<br>RETURN VALUES:<br>outarg[1]: 0 on success, 0 on any error | - |
| playmem | soundmex('playmem', vector, nBits, nSampFreq, Mode, LoopCount, DeviceID)<br>plays sound from memory.<br>PARAMETERS: | sync_play.m |

| | | |
|---|---|---|
| | vector: one or two column vector (mono or stereo)<br>ATTENTION: only values < 1 within vector , otherwise clipping will occur<br>ATTENTION: if playing in raw mode (see 'setmode'), never touch this vector during playback!!<br>nBits: Bitlength for playback!! (16 or 32). Use '32' for 24-bit devices!!<br>nSampFreq: Samplingfrequency for playback.<br>Mode: string ('sync' or 'async')<br>async: starts playback and returns immediately<br>sync: starts playback and returns after playback is complete<br>LoopCount: number of times file vector should be played. 0 does infinite loop (in async mode only)<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| preparemem | soundmex('preparemem', vector, nBits, nSamFreq, Mode, LoopCount, DeviceID)<br>same as playmem, but device is opened only. Use this for multiple devices to synchronize them and start all with 'playprepared'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>ATTENTION: ONLY ASYNC PLAYBACK MODE SUPPORTED!<br>PARAMETERS:<br>same as in playmem<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | sync_play.m |
| memtofile | soundmex('memtofile', vector, nBits, nSampFreq, outfile, DeviceID)<br>pumps vector through DSP-pipe to file.<br>ATTENTION: volumes set by command 'setvolume' are NOT applied!<br>PARAMETERS: | - |

| | | |
|---|---|---|
| | vector: one or two column vector (mono or stereo)<br>ATTENTION: only values < 1 within vector , otherwise clipping will occur<br>nBits: Bitlength for output file (16 or 32). Use '32' for 24-bit devices!!<br>nSampFreq: Samplingfrequency for ouput file.<br>outfile: Name for the output file.<br>DeviceID: optional WaveOutDeviceID, default is 0. Must be set here for plugins!<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| playprepared | soundmex('playprepared')<br>starts playback of all devices already prepared using 'preparefile' or 'preparemem'<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | sync_play.m<br>sync_record.m |
| isplaying | soundmex('isplaying', DeviceID)<br>determines if a device is playing wave data at the moment<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: 1 if specified device is playing, 0 else | calculate_while_playing.m<br>ringbuffer.m |
| playposition | soundmex('playposition', DeviceID)<br>returns the actual playing position in samples<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES: | |

| | | |
|---|---|---|
| | outarg[1]: 1 on success, 0 on any error<br>outarg[2]: actual playing position in samples | |
| mixfile | soundmex('mixfile', filename, Mode, DeviceID)<br>mix file to already playing device.<br>PARAMETERS:<br>filename: the filename! Must be a valid Windows-PCM-Wavefile!<br>Mode: 'add' or 'multiply'<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | mark_buttons.m |
| mixmem | soundmex('mixmem', vector, Mode, DeviceID)<br>mixsignal from meory buffer to already playing device.<br>PARAMETERS:<br>vector: one or two column vector (mono or stereo)<br>ATTENTION: only values < 1, otherwise clipping will occur<br>ATTENTION: if playing in raw mode (see 'setmode'), never touch this vector during playback!!<br>Mode: 'add' or 'multiply'<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | modulate.m<br>see mark_buttons.m for mixfile |
| ismixing | soundmex('ismixing', DeviceID)<br>returns 1 if requested Device is mixing or 0 else.<br>ATTENTION: ismixing will return 1 if a mix is 'pending' (data are mixed actually, but the mixed buffers have not arrived at the device yet, and so the mixed data are not audible yet). Before the mixed output is audible 'mixposition' will return negative values! | mark_buttons.m<br>modulate.m<br>record.m |

| | | |
|---|---|---|
| | PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: 1 if specified device is playing, 0 else | |
| mixposition | soundmex('mixposition', DeviceID)<br>returns the actual mixing position<br>ATTENTION: ismixing will return 1 if a mix is 'pending' (data are mixed actually, but the mixed buffers have not arrived at the device yet, and so the mixed data are not audible yet). Before the mixed output is audible 'mixposition' will return negative values!<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: actual mixing position in samples | record.m |
| stopplay | soundmex('stopplay', DeviceID)<br>stops playback of device.<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | calculate_while_playing.m |
| stopplayall | soundmex('stopplayall')<br>stops playback of all devices.<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | calculate_while_playing.m |
| stopmix | soundmex('stopmix', DeviceID) | |

| | | |
|---|---|---|
| | stops mixing, if mixing is active<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| stopall | soundmex('stopall')<br>stops playback and recording of all devices.<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | record.m |
| hasclipped | soundmex('hasclipped', DeviceID)<br>returns the number of clipped samples since the last 'play' command.<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: number of clipped samples since the last start of the device | calculate_while_playing.m<br>mark_buttons.m |
| setplaymode | soundmex('setplaymode', Mode)<br>sets playback mode.<br>PARAMETERS:<br>Mode: one of the follwing values:<br>raw: fast, but not safe: vector from MatLab is played directly<br>in soundmex Manipulating that vector during playback will result in an access violation!!<br>safe: data from Matlab will be copied in memory. This mode is default<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | rawmode.m |
| getvolume | soundmex('getvolume', DeviceID) | |

| | | |
|---|---|---|
| | returns the actual volume factor of the device.<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: actual factor applid to wave data | |
| setvolume | soundmex('setvolume', Volume, DeviceID)<br>sets a volume factor that is applied to the samples before they are sent to the device.<br>PARAMETERS:<br>Volume: All samples will me multiplied with this value in real-time. Values above 1 or below -1 may cause clipping!<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | calculate_while_playing.m<br>mark_buttons.m<br>modulate.m<br>record.m |

### 6.1.3 Recording Commands

| Command | Description | Examples |
|---|---|---|
| setrecordformat | soundmex('setrecordformat', BitLength, Channels, SamplingFrequency, DeviceID)<br>sets the record format for specified record device<br>PARAMETERS:<br>BitLength: 16 or 32 (default on startup is 16)<br>Channels: 1 or 2 (default on startup is 2)<br>SamplingFrequency: Samplingfrequency to record with (default on startup is 44100)<br>DeviceID: optional WaveOutDeviceID, default is 0 | record.m<br>sync_record.m |

| | | |
|---|---|---|
| | RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| setrecordlength | soundmex('setrecordlength', Length, DeviceID)<br>sets the record time for specified record device. If setting to 0, data are recorded to harddisk until 'stoprecord' or 'stopall' is called.<br>Any other value will record to memory. The last 'Length' samples are stored while recording, older values are overwritten. Therefore 'getrecorddata' will return the last 'Length' recorded samples.<br>PARAMETERS:<br>Length: Length in samples (!)<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | record_length.m |
| record | soundmex('record', DeviceID)<br>starts recording of device.<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | record.m |
| preparerecord | soundmex('preparerecord', DeviceID)<br>same as record but device is opened only. Use this for multiple devices to synchronize them and start all with 'recordprepared'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | sync_record.m |

| | | |
|---|---|---|
| recordthrs | soundmex('recordthrs', DeviceID)<br>starts recording of device after the Threshold was exceeded within the recording device once.<br>PARAMETERS:<br>Threshold: Number between 0 and 1. 0 starts recording immediately<br>NumSamples: Number of samples to record.<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | thrs_record.m |
| preparerecordthrs | soundmex('preparerecordthrs', Threshold, NumSamples, DeviceID)<br>same as recordthrs but device is opened only. Use this for multiple devices to synchronize them and start all with 'recordprepared'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>PARAMETERS:<br>Threshold: Number between 0 and 1. 0 starts recording immediately<br>NumSamples: Number of samples to record.<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| recordprepared | soundmex('recordprepared')<br>starts recording of all devices already prepared using 'preparerecord'<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | sync_record.m |
| isrecording | soundmex('isrecording', DeviceID)<br>determines if a device is recording wave data at the moment<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0 | |

| | | |
|---|---|---|
| | RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: 1 if specified device is recording, 0 else | |
| stoprec | soundmex('stoprec', DeviceID)<br>stops recording of device.<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| stoprecall | soundmex('stoprecall')<br>stops recording of all devices.<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| saverecord | soundmex('saverecord', filename, DeviceID)<br>saves recorded data from Device to file Windows PCM-Wavfile (16 or 32bit integer)<br>PARAMETERS:<br>filename: filename to save wave to<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| getrecorddata | soundmex('getrecorddata', DeviceId)<br>returns recorded wave data from device as row (!) vector<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: row vector | record.m<br>sync_record.m |

### 6.1.4 Input-Output-Commands

| Command | Description | Examples |
|---|---|---|
| startio | soundmex('startio', Waveout-DeviceId, Wavein-DeviceId)<br>starts recording and mapping recorded sound directly to output. Use 'setrecordformat' to set the desired record format before using 'startio'. Playback format will be identical to record format.<br>PARAMETERS:<br>Waveout-DeviceId: WaveOutDeviceID for playback<br>Wavein-DeviceId: WaveInDeviceID for recording<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | start_io.m |
| prepareio | soundmex('prepareio', Waveout-DeviceId, Wavein-DeviceId)<br>same as startio, but devices are opened only. Use this for multiple devices to synchronize them and start all with 'startpreparedio'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>PARAMETERS:<br>same as in startio<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| startpreparedio | soundmex('startpreparedio', Waveout-DeviceId, Wavein-DeviceId)<br>starts io of all devices already prepared with 'prepareio'<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |

### 6.1.5 Device Commands

| Command | Description | Examples |
|---|---|---|
| setplaybuffer | soundmex('setplaybuffer', NumBuffers, BufferSize, DeviceID)<br>Sets buffer number and size of waveout device. USE WITH CARE<br>PARAMETERS:<br>NumBuffers: Number of buffers used by waveout device<br>BuffersSize: Buffersize of buffers used by waveout device<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: number of output buffers after setting<br>outarg[3]: size output buffers after setting | device_info.m |
| getplaybuffer | soundmex('getplaybuffer', DeviceID)<br>returns information about buffer number and size of waveout device in the second, and the size of the buffers in the third return value<br>PARAMETERS:<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: number of output buffers used<br>outarg[3]: size output buffers | device_info.m |
| setrecbuffer | soundmex('setrecbuffer', NumBuffers, BufferSize, DeviceID)<br>Sets buffer number and size of wavein device. USE WITH CARE<br>PARAMETERS:<br>NumBuffers: Number of buffers used by wavein device | see setplaybuffer in device_info.m |

| | | |
|---|---|---|
| | BuffersSize: Buffersize of buffers used by wavein device<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: number of output buffers after setting<br>outarg[3]: size output buffers after setting | |
| getrecbuffer | soundmex('getrecbuffer', DeviceID)<br>returns information about buffer number and size of wavein device in the second, and the size of the buffers in the third return value<br>PARAMETERS:<br>DeviceID: optional WaveInDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: number of output buffers used<br>outarg[3]: size output buffers | see getplaybuffer in device_info.m |
| getdeviceinfo | soundmex('getdeviceinfo')<br>returns information on all installed WaveOut and WaveIn devices<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: string based information on all devices | device_info.m |

### 6.1.6 Ringbuffer Commands

| Command | Description | Examples |
|---|---|---|
| setringbuffersize | soundmex('setringbuffersize', Size, DeviceID)<br>sets size of ringbuffer<br>PARAMETERS:<br>Size: Size in double values, default is 81920<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | ringbuffer.m<br>synchronized_ringbuffers.m |
| setringbuffermode | soundmex('setringbuffermode', Mode, DeviceID)<br>sets mode of ringbuffer<br>PARAMETERS:<br>Mode: string, what to do, if buffer runs empty:<br>error_on_empty: an error returns if bufferunderrun occurs<br>stop_on_empty: device is stopped if bufferunderrun occurs<br>zero_on_empty: zeros are played if bufferunderrun occurs (silent playback)<br>in this mode the device keeps running, you can use<br>putbuffer later on without restarting playback<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | ringbuffer.m<br>synchronized_ringbuffers.m |
| playringbuffer | soundmex('playringbuffer', nBits, nSamplingFrequency, nChannels, DeviceID)<br>starts playback of ringbuffer. Some Buffers must be filled before starting,<br>if Mode is not set to 'zero_on_empty'<br>PARAMETERS: | ringbuffer.m |

| | | |
|---|---|---|
| | nBits: Bitlength for playback!! (16 or 32). **Use '32' for 24-bit devices!!**<br>nSamplingFrequency: Samplingfrequency for playback.<br>nChannels: channels for playback (1 or 2).<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | |
| prepareringbuffer | soundmex('prepareringbuffer', nBits, nSamplingFrequency, nChannels, DeviceID)<br>same as playringbuffer, but device is opened only. Use this for multiple devices to synchronize them and start all with 'playprepared'. DEVICES MUST SUPPORT SYNCHRONIZATION<br>PARAMETERS:<br>same as in playbuffer<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | synchronized_ringbuffers.m |
| putringbuffer | soundmex('putringbuffer', vector, DeviceID)<br>puts data to the ringbuffer<br>PARAMETERS:<br>vector: one or two row vector (mono or stereo)<br>ATTENTION: Here the format is different to playmem (samples in rows, _not_ in columns!!!!<br>ATTENTION: only values < 1, otherwise clipping will occur<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: 1 if buffer data were accepted, 0 if ringbuffer is busy (filled completely).In that case try it again later!! | ringbuffer.m<br>synchronized_ringbuffers.m |

### 6.1.7 Special Commands

| Command | Description | Examples |
|---|---|---|
| tic | soundmex('tic')<br>resets SoundMex internal tic-toc-counter<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | tictoc.m |
| toc | soundmex('toc')<br>returns elapsed time since last 'tic' command in milliseconds<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: elapsed time since last tic command in milliseconds | tictoc.m |
| setbutton | soundmex('setbutton', windowcaption, buttoncaption, mode, startpos, duration, DeviceID)<br>use soundmex to highlight buttons in a Matlab Window<br>PARAMETERS:<br>windowcaption: Caption of the window with the buttons (substring sufficient)<br>buttoncaption: Caption of the button (exact string!)<br>mode: 'mix' or 'org'. Calculate positions relative to original played loop in background or relative to mixfile (mixmem)<br>startpos: Position in samples when button should be highlighted<br>duration: After 'startpos' + 'duration' the button returns to normal state<br>DeviceID: optional WaveOutDeviceID, default is 0<br>CAUTION: if you want to use the 'org' parameter, you _must_ call 'setbutton'<br>before starting the playback, if you use 'mix' you may call it after starting<br>the playback, but before start mixing!!<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | mark_buttons.m |

| | | |
|---|---|---|
| debugsave | soundmex('debugsave', filename, DeviceID)<br>saves wave output to wave file<br>PARAMETERS:<br>filename: filename. Passing an empty string '' as filename disables debugsave<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error | modulate.m |
| sendwaveoutplugin | soundmex('sendwaveoutplugin', command, PluginId, DeviceId)<br>sends a command to a plugin<br>PARAMETERS:<br>command: command to be sent. Plugin or PluginChain itself must recognize command. PluginChain itself only knows commands 'show' and 'hide'. If passing 'show' you can pass additonal parameters in the string separated by blanks:<br>first parameter: left position of window in pixels<br>second parameter: top position of window in pixels<br>the complete rest: caption of window<br>for example:<br>soundmex('sendwaveoutplugin', 'show 10 100 My Window', -1, 0)<br>PluginId: position of plugin within chain (first plugin has ID 0, PluginCain itself has ID -1)<br>DeviceID: optional WaveOutDeviceID, default is 0<br>RETURN VALUES:<br>outarg[1]: 1 on success, 0 on any error<br>outarg[2]: string based return value from plugin (depends on command)<br>See also the chapter The SoundMex realtime DSP-Plugin-Pipe | plugin.m |
| sendwaveinplugin | same as 'sendwavoutplugin' but for a plugin chain defined for a WaveIn-Device | nn |

# 7 The SoundMex realtime DSP-Plugin-Pipe

SoundMex contains a built-in realtime DSP-Plugin-Pipe. It is avaliable only with the DSP and DSP-SDK licences.

### General descriptions

1. **Plugins:**
   SoundMex DSP-Plugins are binary modules (DLL's) that can manipulate sound data sent to or received from a wave device on the fly. Several plugins can be *plugged* together using a plugin manager, a so called *plugin chain.* Various (and a growing number of) generic plugins is shipped with SoundMex, user defined plugins can be developed using the SoundMex-DSP-SDK (Software Development Kit, see below).

2. **Plugin chains:**
   Every device (recording or playback device) can define a so called *plugin chain*. A *plugin chain* is a container for a variable number of SoundMex DSP-Plugins. The number of plugins is limited only by the speed of your computer, and how 'expensive' calculations are within the single plugins. The generic plugins are highly optimized and do not consume much processor power (except for the 'Visualisation' plugin). The plugins are *plugged* together by the chain in a manner, that the output data of the first plugin are the input data for the second plugin and so on.

3. **Activating a plugin chain:**
   A plugin chain for one or more devices is activated by adding a line to the SoundMex INI-File (either the default file, or the user definded file, see command 'init'):

   ```
   [WaveOutPlugins]
   0=plugins/plugin0.ini
   1=plugins/plugin1.ini
   …

   [WaveInPlugins]
   0=plugins/plugin7.ini
   1=plugins/plugin8.ini
   …
   ```

   The sections *WaveOutPlugins* and *WaveInPlugins* respectively denote if the chain is to be plugged into the signal path of a playback or recording device. The name of the value is the corresponding DeviceID of the recording or playback device. The value (right hand of the '=') specifies another INI-File describing a single plugin chain (see below)

4. **Configuring a plugin chain:**
   Every plugin chains is configured with a INI-File. Such a chain INI-File contains an optional section *GlobalData* and one section for each plugin to be used.

   ```
   [GlobalData]
   MyData=100
   MyData2=200

   [0]
   FileName=plugins/visualize.dll
   LogView=1

   [1]
   ```

```
FileName=plugins/olaeq.dll
EQFile=plugins/conv_1024.cnv
OLAFFTLength=1024
OLAWindowLength=800

[2]
FileName=plugins/visualize.dll
LogView=1
```

The *GlobalData* section contains fields that are available to all plugins during initialization (i.e. all plugins can 'read' these values).
All other sections decribe one plugin each. The names of the section must be ascending integer values (not subsequent, but ascending). The only required field is *FileName*, which holds the name of the DLL that will be loaded by the plugin chain. All other parameters are plugin-specific. Every plugin can read these values during initialization. Plugins that require further parameters will fail to initialize if a value is missing.

So the above example describes a plugin chain containing three plugins (for a description of those generic plugins please refer to the *SoundMex DSP-Plugins* documentation):
- at the first position the data are visualized (spectrum, spectrogram...)
- the second plugin is an overlapped add equalizer (spectral manipulation)
- the third is again a visualization (to see changes done by the equalizer)

The data will 'flow' from top to bottom through these plugins.

5. **Communication with plugins:**
After initializing SoundMex you can use the *sendwaveoutplugin* and *sendwaveinplugin* command to communicate with a plugin. String commands can be sent to a single plugin, but the plugin itself must 'recognize' the command. In the example above, you can activate and deactivate the equalizer on the fly using the command

```
soundmex('sendwaveoutplugin', 'active=1', 1, 0); %activation
soundmex('sendwaveoutplugin', 'active=0', 1, 0); %deactivation
```

assuming the plugin chain defined above was activated for waveout device No. 0 (i.e. the file plugino.ini from 'Activating a plugin chain' is used). For a list of recognized commands of the generic plugins please refer to the *SoundMex DSP-Plugins* documentation.

6. **Communication with the plugin chain and updating global data:**
The plugin chain itself can receive commands using -1 as plugin index in the *sendwaveoutplugin* and *sendwaveinplugin* command. The following commands are supported:

| Command | Description |
|---|---|
| show | shows the specified plugin chain. The command takes three optional parameters, separated by spaces: first is the window position in pixels from the left, second is the window position in pixels from the top, and complete rest will be set as window caption, for example:<br>soundmex('sendwaveoutplugin', 'show 100 100 Hello World', -1);<br><br>**Attention:** If you want to use the second or third parameter, all prior arguments must be given as well!<br>Calling show without parameters will show the plugin chain either at the top left corner of the screen, or - if the SoundMex main window is visible - to the |

| | |
|---|---|
| | right of the main window. |
| hide | hides the plugin chain<br>soundmex('sendwaveoutplugin', 'hide', -1); |
| notify | with notify you can 'update' or 'add' global data. All plugins (except external plugins, see SDK) will re-read settings that they have read on initialization (if possible). You can specify an arbitrary number of arguments in the form 'name=value' separated by space, e.g.<br><br>soundmex('sendwaveoutplugin', 'notify MyData=50 MyData2=100', -1);<br><br>Attention: the plugin itself must support notify actions. Please test your application seriously! |

You can show/hide a plugin chain by clicking the small button to the right of the levelmeter on the main SoundMex window after calling soundmex('show'), too.

For an example see 'plugin.m' in the examples directory. For a description of the generic SoundMex DSP-Plugins and their parameters please refer to the *SoundMex DSP-Plugins* documentation.

# 8 The SoundMex DSP-SDK

The SoundMex DSP-SDK is a software development for building user defined realtime plugins for the SoundMex DSP-Plugin-Pipe. It provides two different types of user defined SoundMex DSP-Plugins:

- *plugins for manipulations in the time domain*: the plugin gets blockwise access to the wave data as normalized floating point values and can manipulate each sample directly before it is sent to the device (or a subsequent plugin).
- *plugins for manipulations in the frequency domain*: the plugin gets access to the complex FFT-spectrum of signal blocks before they are reconstructed and sent to the device (or a subsequent plugin). Data segementation before the FFT and data reconstruction after the IFFT is done by overlapped add procedure. FFT-length as well as window length (zero padding if window length ‹ FFT-length) and window feed (half, quarter or eighth) for the overlapped add can be set by the user.

Both plugin types can be realized in two different ways:

- script based MatLab plugins: a user defined MatLab command (script or mex) is called on runtime within the DSP-Pipe. Wave data are passed blockwise to the command. Data can easily be manipulated in the script and passed back to the DSP-Pipe as output argument. This kind of plugin is extremly easy to use, but is limited in the performance.

- ‘binary’ plugins: high performance plugins can be realized in any programming language that can build Windows Dynamic Link Libraries (DLL). These libraries can be written in any programing language that can build Dynamic Link Libraries (DLL). Implementation can be easily done without any knowledge of windows multimedia, sound devices or other 'special' programing skills. Implementing one function with only some ANSI-C floating

point calculations (the manipulation of the signal) is sufficient for many applications. A simple API reference and some examples are shipped with the SDK.

For a complete documentation please refer to the *SoundMex DSP-SDK* documentation that is available on demand.

# 9 Feature Matrix

SoundMex is shipped with different licences with different registration fees. The feature matrix below gives an overview on the different licences. The SDK licences are identical to ProfessionalDSP licences with the additional feature, that they can load external (user defined) plugins.

| | Free | Basic | Professional | ProfessionalDSP | BasicRec | ProfessionalRec | ProfessionalDSPRec |
|---|---|---|---|---|---|---|---|
| Debugsave | | x | x | x | x | x | x |
| exit | x | x | x | x | x | x | x |
| filetofile | | x | x | x | x | x | x |
| getdeviceinfo | x | x | x | x | x | x | x |
| getlasterror | | x | x | x | x | x | x |
| getplaybuffer | x | x | x | x | x | x | x |
| getrecbuffer | | | | | x | x | x |
| getrecorddata | | | | | x | x | x |
| getvolume | | x | x | x | x | x | x |
| hasclipped | x | x | x | x | x | x | x |
| help | x | x | x | x | x | x | x |
| hide | | x | x | x | x | x | x |
| init | x | x | x | x | x | x | x |
| isinitialized | x | x | x | x | x | x | x |
| ismixing | x | x | x | x | x | x | x |
| isplaying | x | x | x | x | x | x | x |
| isrecording | | | | | x | x | x |
| memtofile | | x | x | x | x | x | x |
| mixfile | x | x | x | x | x | x | x |
| mixmem | x | x | x | x | x | x | x |
| mixposition | x | x | x | x | x | x | x |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| playfile | x | x | x | x | x | x | x |
| playmem | x | x | x | x | x | x | x |
| playprepared | | | x | x | | x | x |
| playringbuffer | | | x | x | | x | x |
| preparefile | | | x | x | | x | x |
| prepareio | | | | | | x | x |
| preparemem | | | x | x | | x | x |
| preparerecord | | | | | | x | x |
| preparerecordthrs | | | | | | x | x |
| prepareringbuffer | | | x | x | | x | x |
| putringbuffer | | | x | x | | x | x |
| record | | | | | x | x | x |
| recordthrs | | | | | x | x | x |
| recordprepared | | | | | | x | x |
| sendwaveinplugin | | | | | | | x |
| sendwaveoutplugin | | | | x | | | x |
| saverecord | | | | | x | x | x |
| setbutton | | x | x | x | x | x | x |
| setplaybuffer | x | x | x | x | x | x | x |
| setplaymode | | | x | x | | x | x |
| setrecbuffer | | | | | x | x | x |
| setrecordformat | | | | | x | x | x |
| setringbuffermode | | | x | x | | x | x |
| setringbuffersize | | | x | x | | x | x |
| setvolume | | x | x | x | x | x | x |
| show | | x | x | x | x | x | x |
| showerror | | x | x | x | x | x | x |
| startio | | | | | x | x | x |
| startpreparedio | | | | | | x | x |
| stopall | x | x | x | x | x | x | x |
| stopmix | x | x | x | x | x | x | x |
| stopplay | x | x | x | x | x | x | x |
| stopplayall | x | x | x | x | x | x | x |
| stoprec | | | | | x | x | x |
| stoprecall | | | | | x | x | x |
| tic | | x | x | x | x | x | x |
| toc | | x | x | x | x | x | x |
| version | x | x | x | x | x | x | x |

# 10 Common problems (FAQ)

| Problem | Solution |
|---|---|
| playback or record has dropouts | increase the buffersize and number of buffers using the 'setplaybuffer' and 'setrecordbuffer' commands |
| dropouts or deadlocks when using loops with soundmex commands in it | put asmall pause anywhere in the loop (e.g. pause(0.0001)), otherwise the windows message loop is not processed accurately |
| system hanging and/or matlab crashes when recording | be sure that you do not run SoundMex from a network drive and that the temporaray path of SoundMex does not point to a network drive (see Installation above) |
| when calling 'getrecorddata' errors like<br><br>File reading error, retrying...<br><br>occur. | be sure that you do not run SoundMex from a network drive and that the temporaray path of SoundMex does not point to a network drive (see Installation above) |
| when playing or recording 32bit waves (or setting recordformat to 32bit) you get an error like<br><br>'specified device cannot play requested format'<br><br>unless your device really should be able to play it | enter the control panel and select 'No Sounds' in your 'Sounds and Multimedia' settings: otherwise some Windows messagebox may play 'ding' in 16 bit mode and your soundcard is blocked for higher resolutions at that time! |

# 11 Upcoming Features

Various realtime DSP-Modules, e.g.

- FIR-filtering
- reverberation
- pitchshifting
- delay

# 12 Version History

Version history starting with version 1.13

Version 1.21, Date: 08.06.05

- Initialization timout increased to 10 seconds for slow plugins
- Filter reading for equalizer plugin rewritten for higher efficiency

Version: 1.20, Date: 26.05.05

- Dongle driver problem on some XP machines fixed

Version: 1.18, Date: 26.10.04

- 'getdeviceinfo' now supported by free version
- documentation changed for 24-bit devices and bitlength parameter
- new command 'playposition' to retrieve actual position within playback

Version: 1.17, Date: 26.10.04

- bugfix for 'random' startposition
- recordthrs rewritten: now the the exact numer of samples starting with the first sample with the 'threshold-value' are saved.

Version: 1.16, Date: 28.09.04

- new 'free' version with limited number of accessible commands introduced (see command 'init')
- init command has additional return value containing licence type information
- new commands 'recordthrs' and 'preparerecordthrs' for recording n samples after a threshold of m was reached in the record channel.

Version: 1.15, Date: 13.09.04

- new command 'setrecordlength' introduced